CS325: Analysis of Algorithms, Fall 2020

SOLUTIONS to Practice Assignment 3

Problem 1. For each of the following statements, respond *True*, *False*, or *Unknown*.

(a) If a problem is decidable then it is in P. **False**

(b) For any decision problem there exists an algorithm with exponential running time. **False**

(c) $P = NP$. **Unknown**

(d) All NP-complete problems can be solved in polynomial time. **Unknown**

(e) If there is a reduction from a problem A to Circuit SAT then A is NP-hard. **False**

(f) If problem A can be solved in polynomial time then A is in NP. **True**

(g) If problem A is in NP then it is NP-complete. **False**

(h) If problem A is in NP then there is no polynomial time algorithm for solving A. **False**

Problem 2. A k-CNF formula is a conjunction (AND) of a set of clauses, where each clause is a disjunction (OR) of a set of exactly k literals. For example,

$$(a \vee b \vee c \vee \neg d \vee \neg e) \wedge (\neg a \vee b \vee c \vee \neg x \vee \neg y) \wedge (\neg x \vee y \vee c \vee d \vee a)$$

is a 5-CNF. The k-SAT problem asks if a k-CNF formula is satisfiable. In class we saw that 3-SAT is NP-hard. In contrast, 2-SAT is polynomially solvable, as it is mentioned in GA4.

(a) Show that 4-SAT is NP-Complete (For partial credit in an exam, specify all the statements you need to conclude that 4-SAT is NP-complete even if you cannot prove them).

Pf: For a problem to be in NP-Complete it must (i) be in NP and (ii) be NP-hard. We prove these statements in succession.

Statement (i): 4-SAT $\in$ NP
In order to prove the given statement, we must show that we can verify a certificate of the problem in polynomial time. In other words, given a proposed solution as input, we need to devise a polynomial time algorithm that determines if this is a correct solution to our problem. Note that a proposed solution to 4-SAT is an assignment of boolean values to each literal in our set of clauses.

Given a proposed solution is a correct solution to 4-SAT, we can determine if it is correct by evaluating each clause. We first assign every literal to its boolean value specified by the solution and then we iterate through each clause exactly once and return $FALSE$ if any

clause does not evaluate to true. If we have iterated through every clause, then we return $TRUE$. It is clear we do this in linear time.

Statement (ii): 4-SAT is NP-Hard
To prove that 4-SAT is NP-Hard we can reduce 3-SAT to it. Recall that in 3-SAT we are trying to satisfy a conjunction of clauses each with exactly 3 literals. We will take a single example of a clause in 3-SAT and show how to convert it into a 4-SAT problem.

Let's begin with the following clause in 3-SAT:

$$(a \vee b \vee c)$$

Then, take an arbitrary literal not defined in your set of clauses, say z, and add it to the 3-SAT in the following way:

$$(a \vee b \vee c \vee z) \wedge (a \vee b \vee c \vee \neg z)$$

In order to ensure we have a proper reduction, we need to prove that when and only when the clause from 3-SAT is satisfied, the conjunction above is satisfied as well. Consider the following observations.

Observation 1: If $(a \vee b \vee c)$ evaluates to $TRUE$, then $(a \vee b \vee c \vee z) \wedge (a \vee b \vee c \vee \neg z)$ must as well. This is because one or more of a, b, c must be true. As a corollary, if $(a \vee b \vee c)$ evaluates to $FALSE$ then $(a \vee b \vee c \vee z) \wedge (a \vee b \vee c \vee \neg z)$ must as well. This is because none of a, b, c are true. Given this fact, exactly one of the two clauses in $(a \vee b \vee c \vee z) \wedge (a \vee b \vee c \vee \neg z)$ will be $FALSE$ meaning the evaluation of the conjunction of both clauses is $FALSE$.

Observation 2: If $(a \vee b \vee c \vee z) \wedge (a \vee b \vee c \vee \neg z)$ evaluates to $TRUE$, then $(a \vee b \vee c)$ must as well. Since z and its complement are in both clauses of the conjunction, one or more of a, b, c must also evaluate to $TRUE$. This implies that $(a \vee b \vee c)$ is $TRUE$. As a corollary, if $(a \vee b \vee c \vee z) \wedge (a \vee b \vee c \vee \neg z)$ evaluates to $FALSE$, then $(a \vee b \vee c)$ must as well. Since z and its complement are in both clauses of the conjunction, one of these clauses will always evaluate to $TRUE$. In the other, which we know is $FALSE$, we have $a \vee b \vee c$. Therefore, $(a \vee b \vee c)$ will also evaluate to $FALSE$ (can prove with contradiction).

If we replace each clause in 3-SAT in the manner described above, we have a solution to 4-SAT if and only if we have a solution to 3-SAT. Thus, our reduction is complete and 4-SAT is NP-Complete.

(b) Describe a polynomial time algorithm to solve 1-SAT.

Given a string of conjunction of literals, we would like to prove if its satisfiable in polynomial time.

Begin with the first literal in the clause and assign it the boolean value that would lead it to evaluate as $TRUE$. Continue to do this for each consecutive literal. If we reach a literal that has already been assigned and it requires the other boolean value to evaluate to $TRUE$, output $FALSE$.

If we reach the end of our conjunction, without returning $FALSE$, return $TRUE$.

This is polynomial because we go through each clause in the input exactly once.

Problem 3. Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary graph G, whether G is 3-colorable. Describe and analyze a polynomial-time algorithm that either computes a proper 3-coloring of a given graph or correctly reports that no such coloring exists, using the magic black box as a subroutine. [Hint: The input to the magic black box is a graph. Only a graph. Vertices and edges. Nothing else.]

Solution. Let $\mathcal{M}$ be the magic box, we design a recursive algorithm that uses $\mathcal{M}$ to solve this problem.

There are two base cases: (i) if the graph has at most three vertices, then we color the vertices of the graph with different colors, and (ii) if the graph is complete and it has more than 3 vertices then it is not 3-colorable (Why?).

Suppose, that $G = (V, E)$ is a non-complete graph with $n > 3$ vertices. If $M(G)$ returns false (that G is not 3-colorable), we return that no 3-coloring of G exists. Otherwise, let $u, v \in V$, such that $(u, v) \notin E$; such a pair exists because G is not a complete graph. Now, call $\mathcal{M}(G + (u, v))$, where $G + (u, v)$ is obtained by adding the edge (u, v) to G. If $G + (u, v)$ is 3-colorable, then compute any 3-coloring of $G + (u, v)$, recursively. Observe that this coloring is also a valid coloring for G (Why?). Otherwise, (if $G + (u, v)$ is not 3-colorable), in any valid 3-coloring of G, u and v must receive the same color (Why?). Now, let G' be the graph obtained from G by merging u and v (replacing u and v with a new vertex x that is adjacent to any vertex y that is a neighbor of u or v in G). Since there is a valid coloring of G, in which u and v have the same color, $G' = (V', E')$ is 3-colorable. Now, recursively color G' with three colors a, b, c; let this coloring be $\gamma' : V' \to \{a, b, c\}$. Without loss of generality, assume that x is colored a ($\gamma'(x) = a$). Thus, all the neighbors of x are colored either b or c. Therefore, the following 3-coloring $\gamma : V \to \{a, b, c\}$ of G implied by γ' is valid: each vertex $z \in V$ other than u, v keeps its color from G' that is $\gamma(z) = \gamma'(z)$, and $\gamma(u) = \gamma(v) = \gamma'(x) = a$.